Privacy-Preserving Collaborative Split Learning Framework for Smart Grid Load Forecasting

Asif Iqbal, Prosanta Gope, *Senior Member, IEEE*, and Biplab Sikdar, *Senior Member, IEEE*

***Abstract*—Accurate load forecasting is crucial for energy management, infrastructure planning, and demand-supply balancing. The availability of smart meter data has led to the demand for sensor-based load forecasting. Conventional ML allows training a single global model using data from multiple smart meters requiring data transfer to a central server, raising concerns for network requirements, privacy, and security. To alleviate this issue, we propose a split learning-based framework for load forecasting. We split a deep neural network model into two parts, one for each Grid Station (GS) responsible for an entire neighbourhood's smart meters and the other for the Service Provider (SP). Instead of sharing their data, client smart meters use their respective GSs' model split for forward passes and only share their activations with the GS. Under this framework, each GS is responsible for training a personalized model split for their respective neighbourhoods, whereas the SP can train a single global or personalized model for each GS. Experiments show that the proposed models match or exceed a centrally trained model's performance and generalize well. Privacy is analyzed by assessing information leakage between data and shared activations of the GS model split.**

***Index Terms*—Split learning, load forecasting, transformers, decentralized learning, privacy-preserving, mutual information.**

I. Introduction

ELECTRICITY load forecasting is crucial for energy management systems as it enables planning for power infrastructure upgrades, demand and supply balancing, and power generation scheduling in response to renewable energy fluctuations. Accurate load forecasting can also result in significant cost savings. In 2016, Xcel Energy saved $2.5 million by reducing their load forecasting error from 15.7% to 12.2% [1].

Sensor-based approaches for electricity load forecasting use historical load traces from smart meters and meteorological data to train machine learning (ML) models. ML models for load forecasting can be trained through localized or centralized methods. Localized training entails developing a dedicated model for each smart meter, facilitating client-level load forecasting. Conversely, centralized training involves building a single model using aggregated data from multiple clients to forecast the load for an entire area [2]. However, transferring data directly from clients' premises to a centralized server imposes a heavy communication load and raises significant privacy and security concerns [3]. For instance, high-resolution smart meter data might disclose when someone is at home, their daily routines, and even specific activities. Moreover, their load signatures can identify certain electrical devices or appliances. For example, the use of medical equipment, home security systems, or specialized machinery can be inferred from the load data [4]. Safeguarding this information is of utmost importance, as it protects an individual's privacy and ensures compliance with stringent data regulations, such as the European Union General Data Protection Regulation (GDPR) [5]. In addition, the growing adoption of smart meters renders the practice of training individual models for each customer, whether locally or centrally, increasingly impractical from both computational and financial standpoints.

In order to alleviate these problems, decentralized deep learning methods like *federated learning* (FL) [6] and *split learning* (SL/SplitNN) [7] have been proposed. These methods decouple the requirement of training an ML model on locally/centrally available data by enabling a group of data holders to train an ML model collaboratively without sharing their private data. In FL, a server contains a global ML model shared among multiple clients. During training, each client receives a copy of the global ML model, generates a model update by improving its private data and sends the updated model back to the server. The server then performs aggregation and updates the global model in some way, usually via weighted averaging, and sends the updated model back to each client. In SplitNN, the ML model is split into two parts, one remains on the client's side and the other on the server's side. The client performs the forward pass on its side and shares the outputs with the server, which continues the forward propagation and computes the loss. The gradients are sent back to the client to complete an update step. Clients can choose any ML architecture, as the server has no control over it. In both FL and SplitNN, clients do not share their private data with anyone. These decentralized learning approaches have resulted in a major paradigm shift from an expensive central ML system to utilizing various distributed computational resources.

A. Related Works

This section reviews the recent ML methods proposed for load forecasting, followed by studies that use FL and SL for distributive load forecasting.

Received 11 March 2024; revised 18 April 2025; accepted 28 June 2025. Date of publication 4 July 2025; date of current version 4 November 2025. This work was supported in part by the Asian Institute of Digital Finance (AIDF) under Grant A-0003504-09-00. *(Corresponding author: Prosanta Gope.)*

Asif Iqbal and Biplab Sikdar are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583 (e-mail: aiqbal@nus.edu.sg; bsikdar@nus.edu.sg).

Prosanta Gope is with the Department of Computer Science, The University of Sheffield, S10 2TN Sheffield, U.K. (e-mail: p.gope@sheffield.ac.uk).

1) Load Forecasting: Although the load forecasting problem is not new and several methods have been developed for it [8], we focus on recently proposed deep learning (DL) techniques which have been dominant in sensor-based forecasting [2]. Among these DL architectures, recurrent neural networks (RNN) are well suited for learning the temporal patterns present in smart meter data and have been shown to outperform classical statistical and other ML approaches [2].

Authors in [8] have used the attention mechanism to develop a Sequence to Sequence RNN (S2S RNN) for load forecasting using two RNNs. Their use of an attention mechanism aids in capturing the long-term dependencies present in the load traces by improving the link between both RNNs. In [9], authors used S2S RNN to perform load forecasting for several clients via Similarity Based Chained Transfer Learning (SBCTL), where they train a model for a single client traditionally while the other clients utilize transfer learning to build upon the already trained model. In [10], authors present an online adaptive RNN model to train the model as new data arrives continuously. They use a Bayesian Normalized LSTM (BNLSTM) as their base learner and use an online Bayesian optimizer to update model weights online. Similarly, the authors in [11] propose a multi-layer perceptron mixer structure to perform 24-hour-ahead forecasting.

Following an excellent performance in computer vision (CV) [12] and natural language processing (NLP) community [13], the Transformer architecture [13] has recently been employed to capture long-term dependencies in time-series forecasting problems [14], [15], [16]. Instead of working with a single time point at a time (as in RNN), transformer models perform sequence-to-sequence (instead of one sample ahead) forecasting using an encoder-decoder architecture. At the core of transformers, there are self-attention and cross-attention mechanisms which, in vanilla transformer [13], use a point-wise connected matrix leading to a quadratic computational complexity $\mathcal{O}(N^2)$ w.r.t. the input sequence size.

For the time-series forecasting problem, the quadratic complexity of the vanilla transformer is prohibitive. Thus, various modifications to the attention mechanism have been proposed to reduce its complexity. Authors in [17] employ log-sparse attention to bring the complexity down to $\mathcal{O}(N \log^2 N)$. In [15], Informer architecture is proposed, which uses KL-divergence based ProbSparse self-attention mechanism and a distilling operation to reduce the complexity to $\mathcal{O}(N \log N)$. Authors in [14] propose Autoformer, which replaces the canonical attention with an auto-correlation block to achieve sub-series level attention with $\mathcal{O}(N \log N)$ complexity. In FEDformer [16], similar to [14], authors replace the canonical attention with an attention mechanism implemented in the frequency domain (using FFT or wavelet transform). They perform low-rank approximation in the frequency domain and use the mixture of experts' decomposition to separate short-term and long-term patterns, leading to linear complexity $\mathcal{O}(N)$. Drawing on the comprehensive performance evaluations reported in FEDformer [16] and Autoformer [14], these methods outperform recently proposed transformer-based, LSTM-based, and statistical-based approaches in both univariate and multivariate prediction tasks across a prediction horizon of 4-30 days.

2) Decentralized Learning Methods: In their primary forms, both FL and SL frameworks assume that a single model can capture trends across diverse clients; thus, for the load forecasting application, these naive approaches try to learn a single model capable of generating load traces for each client. This, however, is not optimal as the pattern diversity between the clients is usually large, and learning a single forecast model may lead to inferior performance. When used for load forecasting using smart meter data, the methods reviewed so far learn one individual model for each smart meter client or one for a particular group. However, this training strategy becomes computationally expensive as the number of clients grows and raises privacy and security vulnerabilities associated with centralized data transfer.

Several FL and SL-based methods have been proposed to address these issues. In [19], an FL-based method has been presented for short-term (One-hour) load forecasting for smart meters with similar load profiles. Their approach uses LSTM as the learning model and federated averaging architecture with weighted averaging for model weights aggregation. The method is shown to work well for short-term (one hour ahead) predictions. Similarly, [9] presents a similar approach for short-term forecasting with an emphasis on providing security to the framework via encryption schemes. This, however, leads to increased time complexity of the model. In contrast to [19], authors in [22] compare the performance of two FL techniques, FedSGD (single gradient descent step per client) and FedAVG (multiple gradient updates before merging), and allow their clients to have profiles from different distributions. Similarly, recent studies like [24] and [25] have leveraged RNN, LSTM, and GRU architectures to train global models for short-term forecasting within the FL framework. Furthermore, in the work by [25], differential privacy techniques were applied to obscure signs of shared client gradients, providing an additional layer of protection for client privacy. Similarly, in [28], the authors combine FL with the principles of transfer learning to perform demand side forecasting using a Transformer model. In another work [26], the authors introduce an FL-based boosted multi-task learning framework tailored for inter-district collaborative load forecasting (1 h ahead). The approach revolves around initially training a central model, which is subsequently employed by individual districts to train personalized models capable of capturing their respective local temporal dynamics. A notable feature of this framework is its use of the probabilistic Gradient-Boosted Regression Tree (GBRT) as the base learner.

Moving on to works that leverage for privacy preservation, in [29], authors integrate the FL strategy with local DP to protect user data in recommendation systems. Similarly, in [30], authors enhance user privacy in task-oriented semantic communication within the 6 G landscape by incorporating DP and encryption during the training of a deep neural network-based joint source and channel coding (DeepJSCC) model. In [31], authors use the SL framework to split a 1D CNN network model into two halves and use it to detect heart abnormalities from the medical ECG dataset. Furthermore, they show that in the case of 1D CNN, SL may fail to protect the patients' private raw data. To mitigate this data leakage, they use differential privacy (DP) [32], where carefully computed noise is added to the patients' activations as

TABLE I
SUMMARY OF RELATED WORK ON SMART GRID LOAD FORECASTING

Scheme	Training Framework Used	Deep Learning Architecture Used	Forecast Horizon (hours)	Models Trained	Privacy Preservation Methodology & Analysis		
					Method Applied	Quantitative†	Qualitative‡
Tian *et al.* [9]	SBCTL	S2S RNN	1	Global	★	★	★
Sehova *et al.* [8]	Central	S2S RNN	1, 24	Global	★	★	★
Fekri *et al.* [10]	Central	BNLSTM	1 - 200	Global	★	★	★
Yazici *et al.* [18]	Central	1D-CNN, LSTM, GRU	1, 24	Global	★	★	★
Ryu *et al.* [11]	Central	MLP-Mixer	24	Global	★	★	★
Taik *et al.* [19]	FL	LSTM	1	Global	FL	✗	✗
Li *et al.* [20]	FL	LSTM	1	Global	FL	✗	✗
Liu *et al.* [21]	FL	LSTM	1, 7	Global	FL + HE	-	-
Fekri *et al.* [22]	FL	LSTM	1, 24	Global	FL	✗	✗
Yang *et al.* [23]	FL	SecureBoost	1	Global	FL	✗	✗
Liu *et al.* [24]	FL	RNN + LSTM + GRU	0.5 - 4	Global	FL	✗	✗
Husnoo *et al.* [25]	FL	RNN + LSTM + GRU + CNN	1	Global	FL + DP	✗	✓
Liu *et al.* [26]	FL	Prob-GBRT	1	Global & Local	FL	✗	✗
Sakuma *et al.* [27]	SL	S2S LSTM + GRU + 1D-CNN	1, 24	Global & Local	SL	✗	✗
Proposed	SL	S2S FEDformer	96	Neighbourhood	SL + DP	✓	✓

Training framework abbreviations: *SBCTL*: Similarity based Chained Transfer Learning, *FL*: Federated Learning, *SL*: Split Learning.
Deep learning model abbreviations: *S2S*: Sequence to Sequence, *RNN*: Recurrent Neural Network, *LSTM*: Long Short-Term Memory, *GRU*: Gated Recurrent Unit, *BNLSTM*: Bayesian Normalized LSTM, *CNN*: Convolutional Neural Network, *MLP*: Multi Layer Perceptron, *Prob-GBRT*: Probabilistic Gradient-Boosted Regression Trees.
HE: Homomorphic Encryption, *DP*: Differential Privacy.
★: Privacy was not considered while the central model was trained. ✓: Included. ✗: Not included. - : Not applicable.
† Quantitiative: Similarity analysis between clients' data and shared information.
‡ Qualitative: Effects of privacy preservation approach on model performance.

an additional layer of security. Their results show that DP does reduce privacy leakage but at the expense of model performance. Similar to [31], another recent SL-based method [33] splits an LSTM network to train a classifier for time-series data of multiple patients. To reduce privacy leakage, differential privacy has been used to break the 1-1 relationship between input and its split activations.

A short summary of related work on the smart grid load forecasting problem is given in Table I.

B. Problem Description and Motivation

Consider a scenario where an energy provider company distributes power to multiple neighbourhoods/districts of a city. Each community is served by a single Grid Station (GS). The Service Provider (SP) is interested in training a load forecasting model for medium (few hours) to long-term (few days - weeks) forecasting to better manage their generation capacity and reduce energy waste. To do so, they can employ different strategies, e.g., the SP might want to learn a single prediction model for all districts, which is easier but not optimal as households across neighbourhoods may have different load profile distributions. Thus, training a single model to cover all distributions may lead to significant prediction errors. Conversely, training a single model for every client is cumbersome and infeasible if the number of clients is substantial. Instead of either of these extremes, we propose to learn a single model for each neighbourhood as one would expect clients from the same neighbourhood to have similar load profiles, facilitating the training of an accurate prediction model.

Next, we need to decide on a training framework, i.e., central or decentralized. As data privacy is our top priority, decentralized learning strategies like FL and SL should be employed where the private data never leaves the client's premises. However, as the client-side training has to be performed by a smart meter, the training process's computational and data transfer requirements have to be modest so as not to hinder their main functionalities. The SL framework is selected to ensure this due to its low computational and communications requirements and privacy-preserving nature.

C. Contributions

The major contributions of this paper are as follows:

- We propose a novel SL framework with a dual split strategy, i.e., the network's first split (Split-1) resides at the GS

covering a single neighbourhood, and the second split (Split-2) stays at the SPs' end. To reduce computational load on smart meters, each client is only responsible for performing a forward pass on its private data using their GSs' Split-1 network, computing the loss, and initiating back-propagation. GS is responsible for carrying out back-propagation through the Split-1 model and updating its weights. The models, as a whole, are trained using two alternative strategies: *SplitGlobal*, which trains unique Split-1 models for each neighbourhood and a global Split-2 model shared by all neighbourhoods, and *SplitPersonal*, which trains personalized split models (Split-1 and Split-2) for each neighbourhood. Once the training is complete, the SP will have access to both network splits and can perform individual-level (requiring respective clients' involvement) and neighbourhood-level predictions using the cumulative load trend from the respective neighbourhood GS.

- As our base learner, we utilize the transformer [13] based architecture, called FEDformer [16]. Compared with widely used LSTM, transformers enjoy better performance and can fully utilize the acceleration offered by discrete graphics processing units. Based on our literature review, ours is the first work that uses a transformer architecture to implement split learning for electricity load forecasting. Extensive experiments are conducted to assess the performance of the trained split model against a centrally trained model under multiple scenarios.
- We present a detailed quantitative assessment of the extent of information leakage between clients' private data and their respective Split-1 activations using mutual information-based neural estimation (MINE) [34]. Based on the estimated mutual information (MI) between input and activations, a vigilant client can decide whether the current activations batch is secure enough to be forwarded to the GS or not. For additional privacy, we incorporate differential privacy [32], [35] to further obfuscate the client's Split-1 activations into the model framework and analyze its effects on information leakage and model performance.

The rest of the paper is organized as follows: Section II briefly describes FEDformer, the transformer variant used in this work. We further discuss the concepts of split learning, differential privacy, and mutual information neural estimation. Section III outlines the proposed system model, the FEDformer model split and the SL training framework. Section IV presents experimental evaluations of the proposed framework under different testing scenarios, followed by privacy leakage analysis using mutual information and differential privacy. We conclude the paper in Section V.

II. Preliminaries

In this section, we briefly describe the frequency enhanced attention blocks proposed in FEDformer [16], split learning framework [7], differential privacy for machine learning [32], and mutual information neural estimation [34] for privacy leakage analysis.

A. FEDformer

FEDformer follows the deep decomposition architecture proposed in Autoformer [14], where the input time series is analyzed by decomposing it into a seasonal and a trend-cyclical part. The seasonal part is expected to capture seasonality, whereas the trend-cyclical part is expected to capture the long-term temporal progression of the input. FEDformer uses a series decomposition block with a single or a set of moving average filters (of different sizes) to perform such decomposition. FEDformer implements self-attention mechanisms in the frequency domain using two distinct blocks, a Frequency Enhanced Block (FEB) and a Frequency Enhanced Attention (FEA) Block. Their working is briefly discussed next.

1) Frequency Enhanced Block: In [16], the authors proposed Fourier transform and Wavelet transform to work in the frequency domain. Here we will only focus on the Fourier transform. Let $\mathbf{X} \in \mathbb{R}^{L \times D}$ be the input to the FEB, where L is the sample length, and D is the inner dimension of the model. Query matrix $\mathbf{Q}$ is computed by linearly projecting $\mathbf{X}$ with $\mathbf{W} \in \mathbb{R}^{D \times D}$ as $\mathbf{Q} = \mathbf{X}\,\mathbf{W}$. Next, $\mathbf{Q}$ is transformed from time to frequency domain using Discrete Fourier transform (DFT) to get $\mathcal{Q} \in \mathbb{C}^{L \times D}$. A subset of randomly selected Fourier components is discarded from $\mathcal{Q}$ to get a reduced dimensional matrix $\tilde{\mathcal{Q}} \in \mathbb{C}^{M \times D}$, where $M < L/2$. Finally, the output of FEB is computed as

$$\text{FEB}(\mathbf{X}) = \mathcal{F}^{-1}(\text{ZeroPad}(\tilde{\mathcal{Q}} \bigodot \mathbf{R}) \tag{1}$$

where $\mathbf{R} \in \mathbb{C}^{D \times D \times M}$ is a randomly initialized parametric kernel, and $\bigodot$ is the production operator. The result of $(\tilde{\mathcal{Q}} \bigodot \mathbf{R}) \in \mathbb{C}^{M \times D}$ is then zero-padded to $\mathbb{C}^{L \times D}$ and transformed back into time domain via inverse DFT.

2) Frequency Enhanced Attention Block: The FEA block takes two inputs, the encoder output $\mathbf{X}_{en}$ and $\mathbf{X}_{de}$ from decoder, and generates the query $\mathbf{Q}$ matrix via linearly projecting $\mathbf{X}_{de}$ using weight matrix $\mathbf{W}_q$, whereas the key $\mathbf{K}$, and value $\mathbf{V}$ matrices are generated by projecting $\mathbf{X}_{en}$ using $\mathbf{W}_k$ and $\mathbf{W}_v$, respectively.

The query, key, and value matrices are then transformed from time to frequency domain via DFT followed by random mode selection (as in FEB Section II-A1) to get $\tilde{\mathcal{Q}}, \tilde{\mathcal{K}}, \tilde{\mathcal{V}} \in \mathbb{C}^{M \times D}$. Finally, the output of FEA is computed as

$$\text{FEA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathcal{F}^{-1}(\text{ZeroPad}(\sigma(\tilde{\mathcal{Q}}\,\tilde{\mathcal{K}}^{\top})\tilde{\mathcal{V}})), \tag{2}$$

where σ is the *tanh* activation function.

B. Split Learning

In split learning (SL) [7], a deep neural network (DNN) is split into two halves; clients maintain the first half and the remaining layers are maintained by a server. Consequently, a group of clients are able to train a DNN collaboratively using (but not sharing) their collective data. Additionally, the server performs most of the computational work, reducing the clients' computational requirements. However, this comes at the cost of privacy trade-off, i.e., the output of earlier layers leaks more information about the inputs [36]. Here, choosing a suitable split size is important for expecting data privacy as it has been

shown that for a relatively small client model, an honest-but-curious [37] server can extract the clients' private data accurately just by knowing the client-side model architecture [31]. Thus, it is recommended that in SL, clients should compute more layers, increasing computational load but incurring stronger privacy [31].

During training, clients perform forward passes using their own data up to the final split layer of DNN. These activations are then shared with the server, which continues the forward pass on its DNN split. If the label sharing between clients and servers is enabled, the server can compute the loss itself. Otherwise, it has to send the activations of its final layer back to the client for loss computation. In this case, the gradient backpropagation begins on the client side, and the client feeds the gradient back to the server, which continues backpropagation through its DNN split. Finally, the server shares the gradients at its first layer with the client, who finishes the backpropagation through its DNN split. When more than one client is participating in training, SL adds all clients into a circular queue, whereby each client takes turns using their private data to train with the server. At the end of a training round, the client shares its updated model weights with other clients either through a central server, directly with each other, or via a P2P network.

C. Differential Privacy

One of the most widely used privacy-preserving technologies is Differential Privacy (DP) [32]. Its effectiveness in safeguarding user data privacy has been extensively demonstrated by adding carefully computed noise to the data. The machine learning community has widely used DP to ensure data privacy [35], [38]. Let $\mathcal{X}$ be the input space, $\mathcal{Y}$ the output space, ϵ the privacy budget parameter, $\delta \in o(\frac{1}{n})$ be a non-negative heuristic parameter, n be the number of samples in the dataset, and $\mathcal{M}$ a randomization mechanism. We say that the mechanism $\mathcal{M}: \mathcal{X} \rightarrow \mathcal{Y}$ is $(\epsilon, \delta)-$differentially private ((ϵ, δ)-DP) if, for any neighbouring datasets $\mathbf{X}_1$ and $\mathbf{X}_2$ (differing by a single element) in $\mathcal{X}$, and any output $S \subseteq \mathcal{Y}$, as long as the following probabilities are well-defined, there holds

$$Pr[\mathcal{M}(\mathbf{X}_1) \in S] \leq e^{\epsilon} \times Pr[\mathcal{M}(\mathbf{X}_2) \in S] + \delta. \quad (3)$$

Intuitively, (3) provides an upper bound (e^{ϵ}) on the difference between outputs of the mechanism $\mathcal{M}$ when applied to two neighbouring datasets, where the value of ϵ controls the overall strength of the privacy mechanism and δ accounts for the probability that privacy might be violated [35]. Thus, to ensure stronger privacy protection, both ϵ and δ should be kept low. With $\delta = 0$, the pure $\epsilon-$DP is shown to be much stronger than the $(\epsilon, \delta)-$DP (with $\delta > 0$) in terms of mutual information [39]. The addition of δ in the formulation is to provide a level of plausible deniability, allowing for a small probability (δ) that an individual's data might be exposed or identified by an attacker. While ϵ governs the average privacy loss incurred, δ plays a role in controlling the worst-case privacy loss scenario [38]. Additionally, the $(\epsilon, \delta)-$DP offers the advantage of advanced composition theorems, enabling a substantially greater number of training iterations compared to pure $\epsilon-$DP with the same ϵ. As a result, most recent works in differentially private machine learning have shifted away from $\epsilon-$DP.

Let $f: \mathcal{X} \rightarrow \mathbb{R}$ be a deterministic real-valued function. Then, in order to approximate this function with an $(\epsilon, \delta)-$DP mechanism, noise calibrated with f's sensitivity (s) is added to its output. Here, sensitivity is defined as $s = \max_{\mathbf{X}_1, \mathbf{X}_2} \|f(\mathbf{X}_1) - f(\mathbf{X}_2)\|$. Intuitively, for high sensitivity, it is much easier for an adversary to extract information about the input [32]. The general mechanism that satisfies the DP is defined by

$$\mathcal{M}(\mathbf{X}) \triangleq f(\mathbf{X}) + \eta \quad (4)$$

where η is a random variable from distribution $\mathcal{N}(0, \frac{2s^2}{\epsilon^2}\log(\frac{2}{\delta}))$ under $(\epsilon, \delta)-$DP or $p(\eta) \propto e^{-\epsilon\|\eta\|/s}$ under $\epsilon-$DP [38]. In our case, f is the split model, $\mathbf{X}$ is a clients' private input dataset, sensitivity s is computed across the batch axis of the Split-1 activations tensor, and the noise is added to the clients' batch activations at Split-1 model output. In this way, the noise level is controlled by the sensitivity s, which comes from the data, the probability δ, and the privacy budget ϵ, which can be set according to the privacy requirements, e.g., $\epsilon = 10$ results in a weak privacy guarantee as compared to $\epsilon \approx 0$, which gives the strongest privacy guarantee but makes the data useless. The privacy guarantee of a DP mechanism (4) is that the likelihood of revealing sensitive information about any individual in the input dataset through the algorithm's output is significantly reduced [38]. In our study, we analyze both DP mechanisms in terms of the mutual information leakage between input and output of the clients' split network.

D. Mutual Information Neural Estimation

The clients' split layer activations in SL frameworks are shared with the server. However, these activations may carry enough information about the input that an adversary might be able to precisely reconstruct the original data [40]. Various researchers have utilized noise addition mechanisms offered by (ϵ, δ)-DP as a security measure to mitigate this issue. However, it is difficult to quantify the relationship between the added noise level and the information leakage risk. Mutual information (MI) is commonly used in information theory to assess how much information can be inferred from one random variable (RV) about another. Compared to correlations, MI can capture non-linear statistical dependencies between RVs [34]; however, it is difficult to compute, especially for high-dimensional RVs. In [34], the authors have proposed to compute MI using neural networks using the fact that MI between two IID RVs $\mathbf{X}$ and $\mathbf{Y}$ is equivalent to the Kullback-Leibler Divergence (KLD) between their joint ($\mathbb{P}_{\mathbf{XY}}$) and product of their marginal ($\mathbb{P}_{\mathbf{X}} \otimes \mathbb{P}_{\mathbf{Y}}$) distributions. According to the Donsker-Varadhan representation of KLD [41], the MI between $\mathbf{X}$ and $\mathbf{Y}$ is lower bound by

$$\begin{aligned} I(\mathbf{X}; \mathbf{Y}) &= D_{KL}\left[\mathbb{P}_{\mathbf{XY}} || \mathbb{P}_{\mathbf{X}} \otimes \mathbb{P}_{\mathbf{Y}}\right] \\ &\geq \sup_{T \in \mathcal{T}} \mathbb{E}_{\mathbb{P}_{\mathbf{XY}}}[T] - \log(\mathbb{E}_{\mathbb{P}_{\mathbf{X}} \otimes \mathbb{P}_{\mathbf{Y}}}[e^T]). \end{aligned} \quad (5)$$

Here $\mathcal{T}$ is any class of functions $T: (\mathbf{x}_i, \mathbf{y}_i) \rightarrow \mathbb{R}$ that satisfies the integrability constraints of Donsker-Varadhan theorem. Under this setting, the authors in [34] use a neural network to model

$\mathcal{T}$, which converts the MI problem to a network optimization one, leveraging neural networks' ability to approximate arbitrary complex functions.

Consider an SL framework where $\mathbf{x}$ and $\mathbf{y}$ are the batched inputs and outputs of the split network, and we are interested in finding how much information about $\mathbf{x}$ can be inferred from $\mathbf{y}$. To do so, we need to estimate the MI between them. Ref. [34] says that this MI is lower bounded by (5). Let T be a neural network. Then, the expectations in (5) are empirically estimated by sampling from joint distribution as $(\mathbf{x}, \mathbf{y}) \sim \mathbb{P}_{\mathbf{XY}}$ and from marginals by shuffling $\mathbf{y}$ across the batch axis to get $(\mathbf{x}, \overline{\mathbf{y}})$. In other words, in $(\mathbf{x}, \mathbf{y})$ the input-output relationship is intact, whereas, in $(\mathbf{x}, \overline{\mathbf{y}})$, this relationship has been broken. The network T is trained by maximizing (5). Thus, if MI between a batched $\mathbf{x}$ and $\mathbf{y}$ is large, (5) computed using a trained network T will be high, and vice versa. We demonstrate this effect in detail in Section IV-E1.

III. Proposed Split Learning Framework for Electricity Load Forecasting

In this section, we first discuss the proposed system model, the adversarial model, the FEDformer model split and their internal modules, followed by our proposed split learning framework and its training methodology.

A. System Model

We consider a three-tier system model with four major entities, as shown in Fig. 1. On top, we have the electricity Service Provider; in the middle, we have Grid Stations responsible for distributing electricity to the individual districts/neighbourhoods. The lowest tier comprises smart meter clients (industrial, commercial, or residential) lumped into neighbourhoods. Under this model, the SP is an organization which procures electricity from various providers and is responsible for meeting the energy requirements of all connected GSs. Moreover, the smart meters can connect to their GS using a secure communications protocol, e.g., cellular network or power-line communications. The GSs are connected to SP via a private network or the Internet. The SP can not connect directly with any client and has to go through the GS to communicate with a client. The objective is to learn a DL time series prediction model, using a split learning framework, on all clients' data without compromising the individual clients' privacy. Once trained, the entire model will be accessible to the SP, and the SP can effectively perform medium to long-term load forecasting for any client from any neighbourhood.

B. Adversary Model

In this paper, we assume a modest security environment, i.e., the GS, SP, and the clients are honest-but-curious [31], [37]. Thus, the participants may not try to poison the training process; however, both the GS and SP may collude to infer information about clients' private data as they have access to the entire model and the client-side split layer activations (details in Section III-D). Furthermore, external adversaries or some malicious clients may try to intercept the split layer activations sent by the other clients to GS to extract their private data. Thus, the attack model considered here is the model inversion attack, which aims to extract clients' sensitive data given only their activations. Under this attack, the objective of the adversary is to find a function $\mathcal{G}$ which can infer the client's private data $\mathbf{X}$ from its split activations $\mathbf{A}$ as $\mathbf{X} = \mathcal{G}(\mathbf{A})$. However, in practice, this inference need not be exact, as a close approximation of the client's data is usually enough.

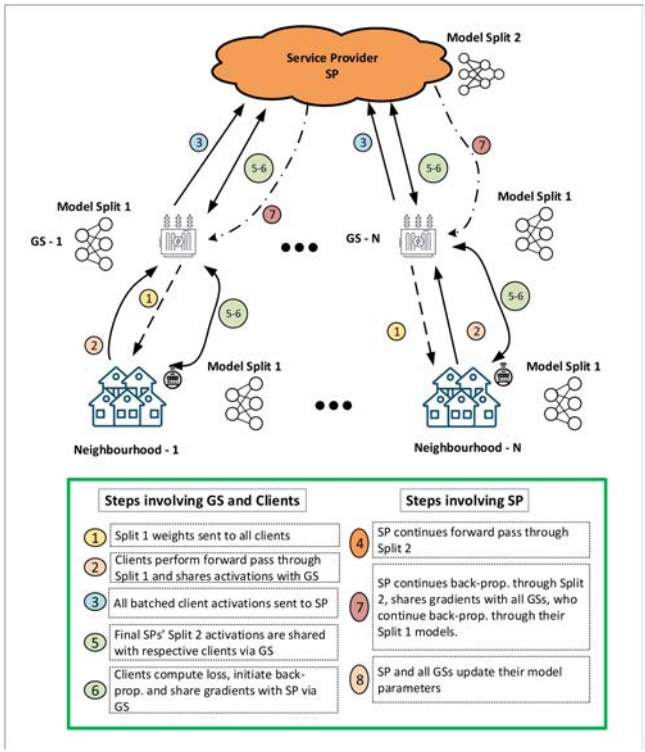


Fig. 1. The proposed system model and split learning framework. SP is the service provider, and GS is the grid station.

C. The FEDformer Model Split

We have selected FEDformer architecture [16] (with some modifications) as our backbone prediction model, where the entire model consists of two encoders and a single decoder block. The FEDformer model is split into two halves, termed Split-1 and Split-2. The Split-1 model contains the first two inner blocks of the FEDformer encoder and FEDformer decoder, whereas the Split-2 model contains the remaining inner blocks of the first FEDformer encoder followed by a complete FEDformer encoder block and remaining inner blocks of the FEDformer decoder. Each GS gets its own copy of the Split-1 model, and the SP maintains Split-2. The resulting split FEDformer is shown in Fig. 2. In Section II-A we have discussed the series decomposition, FEB and FEA blocks, and the rest are briefly discussed next.

The *data embedding layer* included in model Split-1, as shown in Fig. 2, consists of a series decomposition layer, a 1D convolutional layer, and two linear layers. It takes three inputs, an input

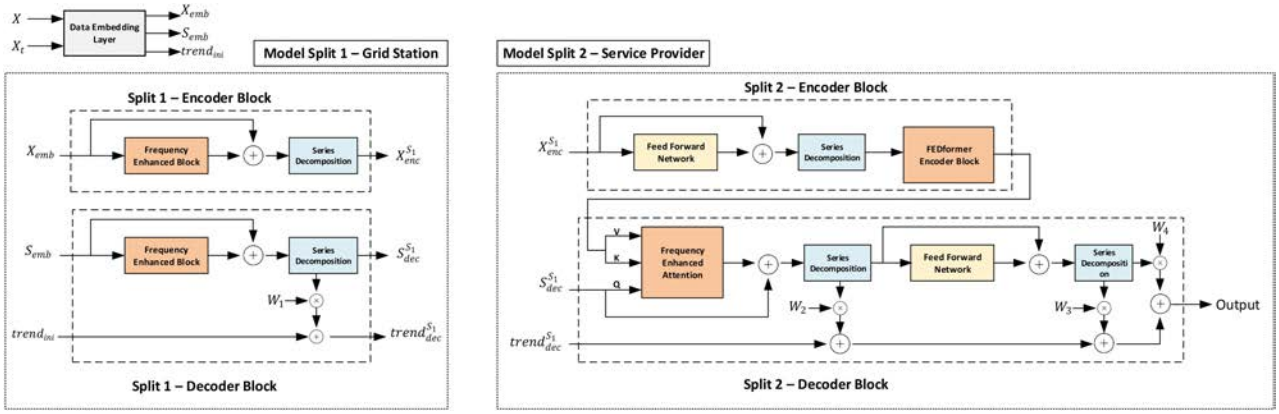


Fig. 2. Dual split FEDformer model.

time-series of length L $\mathbf{X} \in \mathbb{R}^{L \times Z}$, where Z is the dimension of each time point (for univariate case, $Z = 1$), the timestamp encoded information of the input time-series as $\mathbf{X}_t \in \mathbb{R}^{L \times U}$, and output time-series $\mathbf{Y}_t \in \mathbb{R}^{(L/2+O) \times U}$, where U depends upon the temporal granularity, and O denotes the prediction time horizon, for instance, for the scale of Year-Month-Day-Hour, $U = 4$. First, $\mathbf{X}$ is passed through the *series decomposition block* to get a seasonal component $\mathbf{X}_S$ and a trend component $\mathbf{X}_{trend}$. Then, the embedded inputs to the encoder and decoder blocks of Split-1 are generated as:

$$\begin{aligned} \mathbf{X}_{emb} &= 1DConv(\mathbf{X}) + Linear(\mathbf{X}_t) \\ \mathbf{S}_{emb} &= 1DConv(Concat(\mathbf{X}_{S,L/2:L}, \mathbf{X}_0)) + Linear(\mathbf{Y}_t) \\ trend_{ini} &= Concat(\mathbf{X}_{trend,L/2:L}, \mathbf{X}_{mean}), \end{aligned} \tag{6}$$

where $\mathbf{X}_0, \mathbf{X}_{mean} \in \mathbb{R}^{O \times Z}$ denote the placeholders filled with zeros and mean of $\mathbf{X}$, respectively, $\mathbf{X}_{emb} \in \mathbb{R}^{L \times D}$, $\mathbf{S}_{emb} \in \mathbb{R}^{(L/2+O) \times D}$, and $trend_{ini} \in \mathbb{R}^{(L/2+O) \times Z}$. Here, D is the inner dimension of the model. In this way, the decoder takes guidance from the later half of the input time series to fill in the remaining placeholder data points during training. Since the series-wise connection will inherently keep the sequential information, we do not need to perform position embedding, which differs from vanilla Transformers. This specific setting for the Split-1 network was chosen to keep the computational requirements low while ensuring a complex non-linear relationship between the input and outputs of Split-1 to ensure low information leakage between them (see details in Section IV-E1).

The feed-forward network, seen in the Split-2 encoder and decoder, is a two-layer fully connected neural network (FCNN) with input and output dimensions D, and inner dimension D_{ff}. The output of its first layer is passed through *GELU* activation function before passing to layer two. The output (seasonal and trend) tensors generated by series decomposition blocks found in Split-1 and Split-2 decoders have dimensions D, whereas the incoming $trend_{ini}$ and $trend_{dec}^{S_1}$ containing temporal trend information are Z dimensional (dimension of the input time series). Thus, the trend output tensor of a series decomposition block is first projected back into Z dimensional tensors using trainable projection matrices $\mathbf{W}_i \in \mathbb{R}^{D \times Z}$, where $i = [1, 2, 3]$, before adding to the incoming trend tensors. Similarly, the seasonal tensor of the final series decomposition block of the Split-2 decoder is also projected down to Z before adding to the incoming trend tensor to generate the final prediction output.

D. Proposed Split Learning Framework

Fig. 1 depicts the SL training process for load forecasting. Note that the clients do not transfer prediction targets (labels) to the GS or SP. As discussed in Section III-C, the FEDformer is split into two halves, and each GS maintains a personalized copy of model Split-1. The model Split-2 resides at the SP. Unlike a traditional Split Learning framework where clients are responsible for the forward pass, backpropagation through their network split, and parameter updates, our framework proposes a different approach. In our proposed framework, smart meters perform only the forward pass through their data, compute loss (after receiving predictions from SP through GS), and initiate backpropagation (gradient computation of loss with respect to the predictions only). The GS performs backpropagation through Split-1 and parameter updates, significantly reducing the computational load on the smart meters. As a byproduct, we do not need to over-simplify the Split-1 model; thus, non-linear relationships between clients' inputs and activations are maintained, ensuring stronger privacy.

Two strategies are employed when training at SP, i.e., SP can learn a single Split-2 model for all GSs (neighbourhoods) or one personalized Split-2 model per GS. In the case of the former, the overall learned model is referred to as *SplitGlobal*, whereas the latter model is called *SplitPersonal*. The goal is to analyze the generalization capabilities of the model when performing predictions for clients from the same as well as those coming from different neighbourhoods, as clients from different neighbourhoods are expected to have different load patterns and distributions. Nevertheless, the training process is relatively similar for both models. Algorithm 1 outlines the pseudo-code for training the *SplitGlobal* variant of the proposed SL model.

We start with weight initialization for all SP and GS models. At the start of each training epoch, A GS selects all or a subset C_t of K neighbourhood clients (chosen randomly or the same as the previous epoch). Following this, the training begins in parallel across all GSs, where the participating clients use a single private training batch to perform the model update in 7 distinct sequential steps (see Fig. 1). These steps are summarized in relation to Algorithm 1 next.

Steps 1-2: Each GS sends its Split-1 model weights to all selected neighbourhood clients C_t. In parallel, each receiving client k performs a forward pass through the received Split-1 model using one of its private training batches b. The activations of the final split layer, denoted by $\mathbf{A}_{k,b}^{GS}$ of Split-1 model are forwarded to the GS (lines 8-11 of Algorithm 1).

Step 3: Once a GS has received batched activations from all of its training clients, it concatenates their activations, denoted by $\mathbf{A}_{b}^{GS}$, and forwards them to the SP to continue the forward pass (lines 12-13 of Algorithm 1).

Step 4: Depending upon the model being trained, i.e., *SplitGlobal* or *SplitPersonal*, this step is performed little differently. In the case of *SplitGlobal*, the SP performs the forward pass through a single Split-2 model using a batch size of $K \times (clients\, batch\, size)$, where K denotes the number of clients (lines 14-15 of Algorithm 1). For *SplitPersonal*, the SP performs a forward pass through the individual personalized Split-2 models using the activations received from their respective GSs.

Step 5: As the training data is never allowed to leave the client's premises, the final Split-2 layer outputs are forwarded to their respective clients through GSs for loss computation (lines 17-19 of Algorithm 1).

Step 6: Each client computes loss, initiates back-propagation and shares gradients w.r.t. outputs $\mathbf{O}_{k,b}^{GS}$ with their GS, which forwards these gradients to SP (lines 20-22 of Algorithm 1).

Step 7: Once gradients from all GSs are received, SP continues the gradient back-propagation through its Split-2 model(s). The gradients w.r.t. each $\mathbf{A}_{b}^{GS}$ are then forwarded back to their respective GSs. Each GS averages the received gradients across clients' dimensions and continues back-propagating through their respective Split-1 models (lines 24-25 of Algorithm 1).

Step 8: Once all gradients have been populated, SP and each GS update their model weights (line 26 of Algorithm 1).

Following the model updates, the GSs pass their updated models back to their respective clients for another round of batch training. This is repeated until all batches have been iterated over, thus completing a single training epoch. For the next epoch, GSs can continue training with the same clients or select new ones. Once training is finished, each GS will have a personalized Split-1 model.

Algorithm 1: Pseudocode for *SplitGlobal* Model Training Under the Proposed SL Framework.

Input: Number of GSs: nGS, Client Input: $\mathbf{X}$, Client Output: $\mathbf{Y}$, loss function $\mathcal{L}$.

1 **Initialization:** Initialize model weights for *Split-1* and *Split-2* $\forall$ GS and SP models.
2 **for** *t in epochs* **do**
3 Each GS selects $C_t \leftarrow$ set of K clients.
4 Let $nB = \min(nB_1, nB_2, \ldots, nB_K)$, where nB_k is the number of batches available at client k.
5 **for** *b in nB* **do**
6 **Forward Pass:**
7 **for** *GS in nGS* ***in parallel*** **do**
8 GS shares *Split-1* model weights with each client in C_t,
9 **for** *k in* C_t ***in parallel*** **do**
10 $\mathbf{A}_{k,b}^{GS} \leftarrow Split\text{-}1(\mathbf{X}_{k,b}^{GS})$,
11 $\mathbf{A}_{k,b}^{GS}$is shared with respective GS.
12 $\mathbf{A}_{b}^{GS} = Concat(\mathbf{A}_{1,b}^{GS}, \ldots, \mathbf{A}_{K,b}^{GS})$
13 $\mathbf{A}_{b}^{GS}$ is passed to SP.
14 For each GS, SP generates and shares:
15 $\mathbf{O}_{b}^{GS} \leftarrow Split\text{-}2(\mathbf{A}_{b}^{GS})$.
16 **Loss Computation:**
17 **for** *GS in nGS* ***in parallel*** **do**
18 **for** *k in* C_t ***in parallel*** **do**
19 From $\mathbf{O}_{b}^{GS}$, GS sends respective outputs batch $\mathbf{O}_{k,b}^{GS}$ to each client k,
20 $\ell_{k,b}^{GS} = \mathcal{L}(\mathbf{Y}_{k,b}^{GS} - \mathbf{O}_{k,b}^{GS})$,
21 Each client initiates Back-prop. and shares gradients with their GS.
22 GS shares the received gradients with SP.
23 **Back-prop. & Model Update:**
24 SP continues back-prop. through *Split-2* and shares gradients of first *Split-2* layer w.r.t. $\mathbf{A}_{b}^{GS}$ with respective GSs.
25 Each GS resumes back-prop. through their *Split-1* model.
26 SP and each GS perform model update.

Output: Each GS gets a trained *Split-1* model and SP gets trained *Split-2* model.

IV. Experimental Evaluation

In this section, we provide a detailed experimental evaluation of the proposed framework under multiple testing scenarios and present the mutual information-based privacy leakage analysis resulting from sharing activations with and without differential privacy.

1) Dataset and Evaluation Metrics: We used the *Electricity*[1] dataset [14][2] for performance evaluation, which includes hourly electricity consumption of 320 smart meters from July 2016 - July 2019. We selected the first 17,566 entries from July 2016 to July 2018 for training to minimize training time. After normalizing each client's time series to zero mean and unit variance, we used the agglomerative clustering algorithm to group them into three clusters, consisting of 54, 201, and 65 clients, respectively. For visualization, five randomly selected

[1]https://tinyurl.com/fxbcbufp
[2]https://github.com/thuml/Autoformer

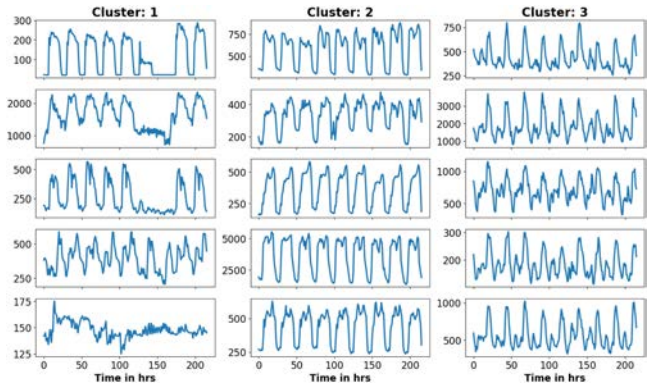


Fig. 3. Five example time-series from each cluster of the *Electricity* dataset.

examples from each cluster over 9 days of data are shown in Fig. 3. Cluster 1 had the most diverse electricity usage patterns, with load patterns and magnitudes varying significantly across clients in each cluster.

The objective of the proposed framework is thus to learn DL models which can accurately perform predictions for clients from each of the three neighbourhoods. We divide each client's load time series into train-val-test sets for training and evaluation purposes according to a 7:1:2 ratio. The evaluation metrics selected are the mean absolute error (MAE), mean square error (MSE), and the coefficient of determination (R^2) between the true and predicted trajectories. The R^2 metric highlights the goodness of fit of a prediction model and is computed as:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - E[y_i])^2}, \tag{7}$$

where y_i is the true time series and $\hat{y}_i$ is the model prediction.

2) Implementation Detail: All of our experiments were performed on Python 3.8, and model implementations were forked from FEDformers' GitHub implementation.[3] We keep the default parameters from FEDformer unchanged unless specified otherwise. Models were trained using ADAM optimizer with adaptive learning rates starting at 10^{-4} and a batch size of 32, with MSE as our training loss. Training is performed over 10 epochs, and an early stopping counter of 3 epochs is used to stop the training once the error over the validation set stops to improve. The input sequence length is set at $L = 96$ hours, output/prediction horizon is set to $O = 96$ hours, model inner dimension $D = 512$, and 64 randomly selected modes are used in FED and FEA blocks. Both FED and FEA are implemented with $h = 8$ attention heads. We choose $K = 10$ clients per neighbourhood in 3 neighbourhoods, with one GS assigned to each neighbourhood. Experiments are repeated 3 times and average results are reported. The split FEDformer contains 2 encoder and 1 decoder layers (see Fig. 2).

All DL models are implemented in PyTorch v1.9 [42] and training is performed on a single Nvidia Tesla V100-32 GB GPU available through a volta-GPU cluster of the NUS-HPC system.[4]

[3]https://github.com/MAZiqing/FEDformer
[4]https://nusit.nus.edu.sg/hpc/

To implement differential privacy, we used the open-source *Diffprivlib* library [43]. The implementation code is publicly available at [44].

A. Comparison With Centralized and FL Models

In our first experiment, we compare the performance of models trained using the proposed SL framework with the FEDformer model trained both centrally and under an FL setting. While FL involves training the entire model on edge clients, in our case, smart meters, this is impractical due to their limited computational capabilities. Nonetheless, we include this comparison for a comprehensive evaluation alongside our approach and the centrally trained model.

To do so, we train *SplitGlobal* and *SplitPersonal* models for 3 GSs and 10 clients per GS. The clients are kept fixed during all training epochs. For the centralized model, we choose the same 10 clients from each GS (30 clients total), as used by our proposed models to train a univariate FEDformer model in a centralized manner. The same strategy is used to train a single model in collaboration with 30 clients under FedAvg scheme as well. The objective is not only to validate our proposed SL training framework but also to assess the efficacy of training multiple models w.r.t. the single model under central and FL training strategies. The resulting MAE, MSE, and R^2 scores on the test sets of clients belonging to different neighbourhoods are summarized in Table II. The MAE, MSE, and R^2 metric scores for GS 2 and 3 for all 3 models are relatively similar, with the Centrally trained model edging the proposed models slightly in terms of MAE, whereas, in terms of MSE and R^2, the *SplitPersonal* model performing better. These results show that all 3 models could generalize the load patterns for GS 2 and 3's neighbourhood clients. The load profiles of these clients were not too erratic compared to clients belonging to neighbourhood 1, for whom the centrally trained model performed objectively worse.

As discussed in Section IV-1, the load profiles for clients belonging to cluster 1 (GS 1) show high diversity and are the most challenging out of the three. Observing the superior performance of the proposed models as compared to the central model shows that expecting a single central model to perform well for a diverse range of load profiles is not viable. Moreover, learning multiple models for data from similar distributions should work comparatively well. Thus, the proposed framework essentially offers a balanced approach between learning a single model for all clients and learning a single model for each client.

When comparing scores of *SplitGlobal* and *SplitPersonal*, we see that the latter performs well both in terms of individual GS scores and average ones. This, however, is expected as for *SplitPersonal*, the SP trains personalized Split-2 networks for each GS. Thus it should be able to generalize well for the respective neighbourhoods. To visualize the model convergence, we present the MSE scores for train, validation, and test sets for the 3 models in Fig. 4. Here, we see that although the central model achieved the lowest training error, its validation and test set errors are the highest, owing to over-fitting to the training

TABLE II
TEST SCORES FOR *SPLITGLOBAL* (SL-G), *SPLITPERSONAL* (SL-P), CENTRAL, AND FEDAVG MODELS FOR DIFFERENT GSS' NEIGHBOURHOODS

	MAE				MSE				R^2			
	SL-G	SL-P	Central	FedAvg	SL-G	SL-P	Central	FedAvg	SL-G	SL-P	Central	FedAvg
GS 1	**0.451**	0.453	0.513	0.535	0.387	**0.383**	0.540	0.554	0.396	**0.405**	0.346	0.342
GS 2	0.256	0.253	**0.246**	0.268	0.133	**0.130**	0.133	0.142	0.863	**0.868**	0.855	0.838
GS 3	0.372	0.361	**0.355**	0.359	0.256	**0.241**	0.252	0.273	0.663	**0.687**	0.671	0.641
Mean	0.360	**0.356**	0.371	0.387	0.259	**0.251**	0.308	0.323	0.641	**0.653**	0.624	0.607

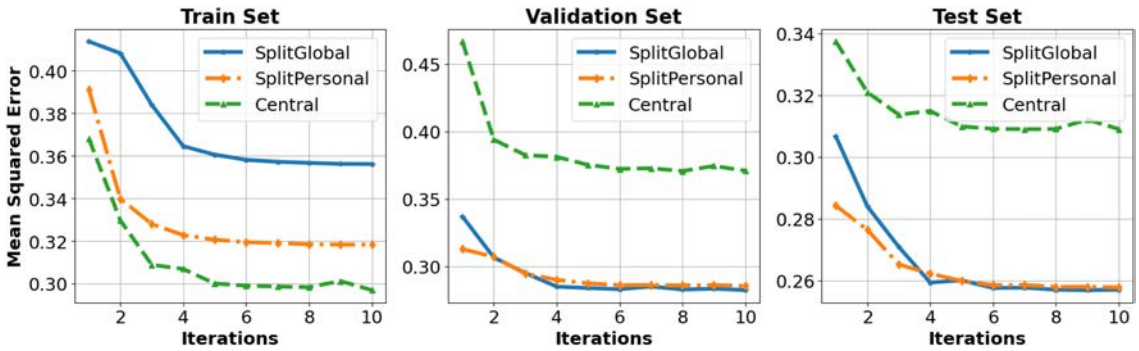


Fig. 4. The train, validation, and test set convergence graphs for *SplitGlobal*, *SplitPersonal*, and Central models.

TABLE III
GS MODEL TEST SCORES ON CLIENTS FROM ALL GS NEIGHBOURHOODS

	SplitGlobal - MSE				SplitPersonal - MSE			
	1	2	3	Mean	1	2	3	Mean
GS 1	0.390	0.249	0.353	**0.331**	0.385	0.379	0.434	0.399
GS 2	0.442	0.134	0.343	**0.306**	0.522	0.130	0.345	0.332
GS 3	0.434	0.186	0.259	0.293	0.426	0.204	0.240	**0.290**
Mean	**0.422**	**0.190**	**0.319**		0.445	0.238	0.340	

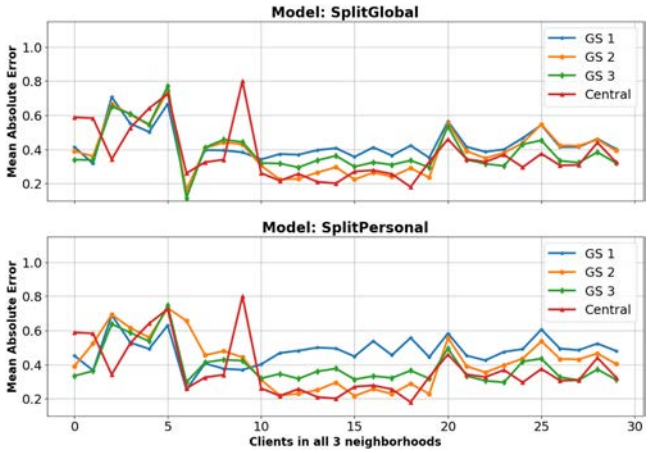


Fig. 5. The test prediction errors of every client for *SplitGlobal*, *SplitPersonal*, and Central models.

data. We can say this because most, if not all, of the difference between central and proposed methods test set scores, is coming from its prediction scores for the GS 1 s' load profiles (see MSE scores given in Table II). As otherwise, the test scores for the central model over GS 1 and 2 were very close to the proposed model's scores. The run-times of a single epoch for *SplitGlobal*, *SplitPersonal*, Central and FL models are 20, 20, 40, and 28 minutes respectively.

B. Across Neighbourhood Predictions

In our next experiment, we analyze the trained models' prediction ability when the tested data comes from another GSs' neighbourhood, i.e., from a different distribution. The MSE scores for both models are reported in Table III, where rows GS 1-3 denote the trained split models and columns denote the neighbourhoods. Thus, the table cell for row GS 1 and column 1 represent the MSE score for neighbourhood 1's test data when GS 1's model is used for prediction. Consider GS 1 models' scores for all neighbourhoods (top row) under both SP training strategies. We see that, apart from testing scores for their own neighbourhood data, the *SplitGlobal* models' prediction errors for neighbourhoods 2 and 3 are well below those reported by *SplitPersonal* model (see Fig. 5 for individual scores). This is attributed to the fact that under *SplitGlobal*, the SP learns a single global Split-2 model, which is jointly trained to learn from all neighbourhood clients. Whereas, under *SplitPersonal* training strategy, the SP learns 3 personalized Split-2 models, which have never seen the data coming from different neighbourhoods. Similar trends can be observed for models GS 2 and 3 where cross-neighbourhood scores reported by *SplitGlobal* are better. Similarly, the mean scores for a single neighbourhood overall GS 1-3 models also show that *SplitGlobal* has better generalization capabilities as compared to *SplitPersonal*.

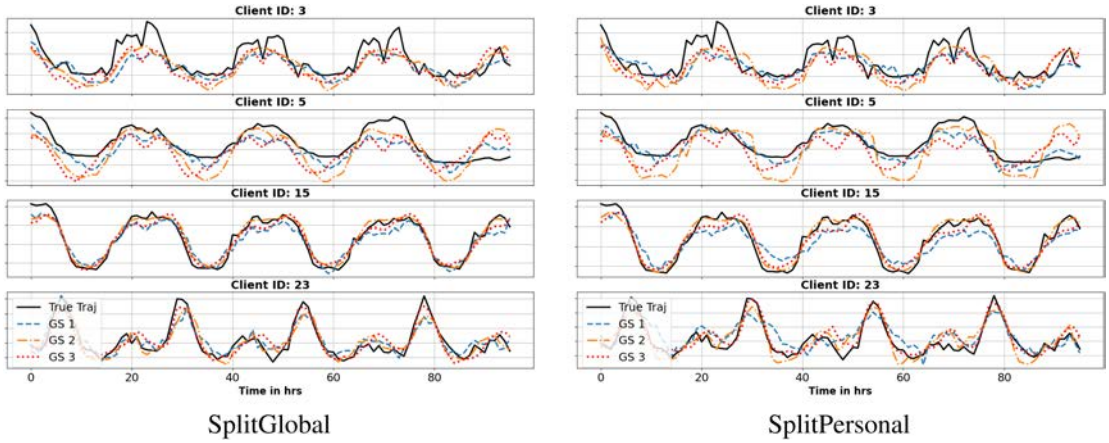


Fig. 6. A single batch prediction for 4 different clients generated by *SplitGlobal* and *SplitPersonal* over the prediction horizon of 96 hours (4 days).

TABLE IV
TEST MSE SCORES FOR SL-G AND SL-P ON SEEN AND UNSEEN DATA

	Same Clients		Random Clients		Post Add. Tr.	
	SL-G	SL-P	SL-G	SL-P	SL-G	SL-P
GS 1	0.390	0.385	0.650	0.644	0.528	0.511
GS 2	0.134	0.130	0.166	0.161	0.154	0.147
GS 3	0.259	0.240	0.218	0.295	0.214	0.264
Mean	0.261	0.252	0.345	0.367	0.299	0.308

TABLE V
TEST MSE SCORES FOR RANDOM CLIENTS WITH MODELS TRAINED USING SAME VERSUS RANDOM CLIENTS IN EVERY TRAINING EPOCH

	Same Clients		Random Clients	
	SL-G	SL-P	SL-G	SL-P
GS 1	0.650	0.644	0.638	0.551
GS 2	0.166	0.161	0.164	0.157
GS 3	0.218	0.295	0.224	0.294
Mean	0.345	0.367	0.342	0.334

To visualize individual test scores for each neighbourhood client under different learned models, we present their MAE scores in Fig. 5. Here, the first 10 clients are from neighbourhood 1 and so on. For all 3 models, we see that the most MAE variation is found in neighbourhood 1's clients, whereas for 2nd and 3 rd neighbourhoods, the variations are small. Moreover, for each neighbourhood, their respective GS models perform best. We further present example batch predictions for 4 clients in Fig. 6. Looking at predictions from both models, we see that for clients 15 and 23, all three GS models do follow the true trajectory very closely, with *SplitGlobal*'s cross neighbourhood predictions being slightly better. However, for clients 3 and 5, belonging to neighbourhood 1, discrepancies are large, especially for cross-neighbourhood predictions of client 5. Moreover, *SplitPersonal*'s GS 1 model followed the true trajectory much more closely, especially near the prediction endpoints.

C. Predictions on Unseen Data

In this experiment, we test the trained models' prediction abilities for clients that are completely new to them. Under usual circumstances, we might not be able to use every client for training; thus, the trained models' generalization capabilities need to be tested. To do so, we start with the models trained on the same 10 clients per neighbourhood and test them on 10 randomly selected clients from each neighbourhood. The results of this test are given in Table IV, where we see that clients coming from 2nd and 3rd neighbourhoods receive slightly worse prediction errors, whereas for neighbourhood 1, the error increases by almost 66%. This performance loss, again, can be attributed to the data diversity found in clients from Neighbourhood 1. To investigate whether further training using these clients brings improvement, we train the model for 5 additional epochs using the selected random clients, whose testing results are summarized in the last two columns of Table IV. We see that additional training not only reduced the error for neighbourhood 1 from 66% to 35%, but also improved scores for the rest of the neighbourhood clients.

D. Training With Random Clients

In this experiment, we compare the prediction performance of models when trained using the same clients (10 per GS) compared to randomly selected clients at every training epoch. The models were trained for 10 epochs and tested against randomly chosen clients, and their results are shown in Table V. Comparing *SplitPersonal*s' GS 1 performance for both training schemes; we see that scores for the model trained using random clients are significantly lower than those trained using the same clients. This could be attributed to the fact that in the former case, the model sees several unique clients during the training stage and is thus able to generalize well for unseen data. Apart from this case, the remaining scores are very similar across the two training schemes. From this observation, we can conclude that training using randomly chosen clients every epoch should enable the models to perform well when presented with unseen data. However, as discussed in the previous section, performing a

few training iterations using the unseen data should be performed to get improved results.

E. Privacy Preservation Using Differential Privacy

Under the proposed framework, both network splits are trained outside the clients' premises, and the training is fully controlled by GS and SP entities. Under the honest but curious security assumption, GS and SP shall not deviate from the training process. However, they may try to infer clients' private data from the received Split-1 activations. In order to secure clients' private data, we propose to use (ϵ, δ)-DP [32], [38] to safeguard against such inference attacks. However, before we do this, we first analyze information leakage between the input and output of the proposed Split-1 FEDformer model. To this end, based on the discussion presented in Section II-D, we perform MI estimation using a fully connected neural network (FCNN). A similar strategy has also been used in [45] to infer MI between inputs and gradients of an NN under the FL framework.

1) Information Leakage Analysis: We use a 3-layer FCNN as our MI neural estimator (MINE), with layers containing 100, 50, and 1 neuron, respectively. The first and second layers use the exponential linear unit (ELU) as their activation function. The loss function, given in (5), is maximized using ADAM optimizer with a learning rate of 10^{-3}. The batch size is kept at 100, and the model was trained over 10^4 epochs. Consider an input tensor for Split-1 model $X_{Inp} = concat(X, X_t)$ (see Fig. 2) of size $nB \times L \times 5$, where $nB = 32$ is the batch size, $L = 96$ is the input sequence length, and each time points consists of 1 load value and 4-dimensional date-time encoded vector. The output tensor of Split-1 Encoder block $X_{Out} = X_{enc}^{S_1}$ is of size $nB \times L \times D$, where $D = 512$ is the inner model dimension. We aim to find the mutual information leakage between X_{Inp} and X_{Out} for a fully trained Split network.

To establish a baseline, we first train the MINE to find the MI between X_{Inp} and itself using a randomly selected client from neighbourhood 2. In order to reduce the computational time, MI is estimated for every 10th batch. The mean and spread over one std of MI computed for each batch is shown in Fig. 7 with approximately a final mean MI value 9.12. The spread seen above and below the mean line signifies the variations of computed MI values over different example batches. With our experimental setting, we can say that the maximum MI between two variables can be at most 9 on average. Having found the upper limit on MI, we next approximate MI between input and a noise tensor of size equal to X_{Out} to establish a lower limit. The entries of noise tensor were generated from Laplace distribution $\mathcal{L}(b = 2)$. The MI score trend for this setting (input - noise only) is also given in Fig. 7 with a final mean value of approx. 1.83. Next, we approximate MI between input and clean X_{Out} as well as $\mathcal{L}(b = 2)$ noise-contaminated $\tilde{X}_{Out}$ and plot them in the same Fig. 7 as well. The final MI for clean and noisy outputs were approx. 3.32 and 2.6, respectively.

Based on the MI approximations discussed above, we see that the MI between input and clean and noisy outputs has a large difference due to the non-linear relationship induced by the Split-1 network. Moreover, the MI of the clean output is much closer to the noise-only case, signifying that the Split-1 network results in minimal information leakage, making the inference attacks much harder to execute [45]. Additionally, as the MI can be computed at the client's end, the client can make an informed decision as to whether current batched activations are safe enough for sharing with the GS.

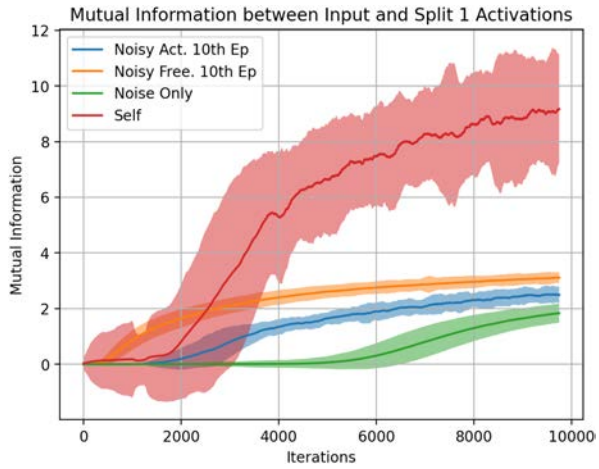


Fig. 7. The mutual information between input X_{Inp}, versus itself, Laplace noise only, clean Split-1 activations X_{Out}, and X_{Out} plus Laplace noise $\mathcal{L}(b = 2)$. The shown spread signifies one standard deviation.

2) Analysis Under $(\epsilon, \delta = 0)-$DP: In this section, we analyze the performance impact of $(\epsilon, \delta = 0)-$DP (pure DP) on model training and testing. To do so, we select a single client from neighbourhood 2 and train the split model under *SplitPersonal* setting. However, this time, the client uses the Laplace mechanism [38], [43] to apply DP to its layer activations (both $X_{enc}^{S_1}$ and $S_{dec}^{S_1}$, see Fig. 2) before forwarding to the GS. The level of noise added is controlled by the privacy budget parameter ϵ, where the noise level is inversely proportional to ϵ. Thus, a lower ϵ results in strong privacy guarantees as compared to higher ones.

To analyze the effects of various privacy budgets, we trained the model with $\epsilon \in [0.5, 1.0, 2.5, 5, 7.5, 10]$ and computed the test scores for the models' predictions. At the 10th training epoch, we also trained our MINE for every 10th batch to approximate the MI between input and Split-1 layer activations. The MI trends for multiple ϵ are shown in Fig. 8, and their respective final MI values, as well as error metrics, are presented in Fig. 9. From Fig. 8 we see that the model trained with $\epsilon = 0.5$ has the lowest approximated MI at 1.86, which is very close to the MI between input and noise only case seen in Fig. 7. As a result of large noise additions, its respective MAE and MSE scores are the worst. However, in terms of MAE, they are only 36% higher than MAE of the non-DP case. Furthermore, increasing ϵ leads to an increase in MI. However, for $\epsilon \geq 5.0$, the MI stagnates and stays very close to 3.3, which is the MI of the non-DP case. In this range, the error metrics are within a margin of error to the non-DP case. This shows that the model can handle DP noise for a low ϵ of 5.0. Even for an $\epsilon = 2.5$ (medium privacy), the MAE and MSE are only 16% and 26% above the non-DP case, showing that good privacy protection can be achieved with mild performance reduction.

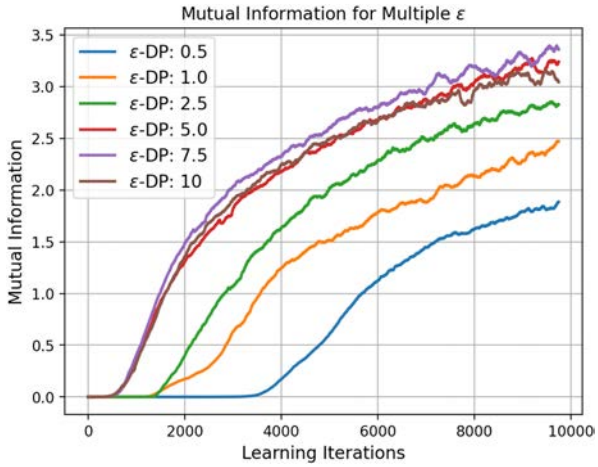


Fig. 8. The mutual information learning trend between input and $(\epsilon, \delta = 0)$-DP protected outputs over a range of ϵ values.

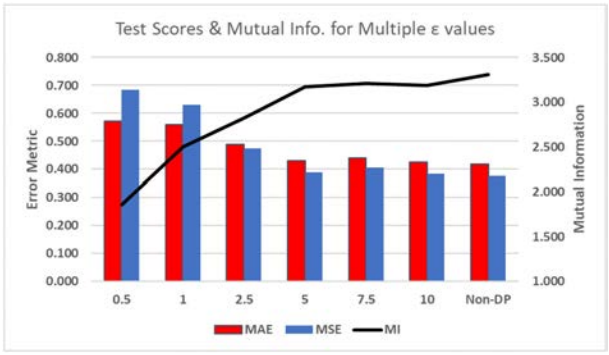


Fig. 9. The prediction test scores and mutual information values for models trained with $(\epsilon, \delta = 0)$-DP over a range of ϵ values.

3) Analysis Under $(\epsilon, \delta)-DP$: Compared to the stricter pure DP [39], (ϵ, δ)-DP introduces an extra parameter, δ, into the framework. The reason behind this addition is to provide a level of plausible deniability, allowing for a small probability (δ) that an individual's data might be exposed or identified by an attacker. While ϵ governs the average privacy loss incurred, δ plays a role in controlling the worst-case privacy loss scenario. Another advantage of $(\epsilon, \delta)-$differential privacy is its advanced composition theorems, enabling significantly more training iterations than pure DP under the same ϵ. This is the reason why most related works in differentially private machine learning have shifted away from pure differential privacy.

To investigate the impact of (ϵ, δ)-DP on MI leakage and model performance, we repeat the previous experiment but with the (ϵ, δ)-DP via Gaussian mechanism [43]. In this experiment, we maintained ϵ at a fixed value of 1.0 and systematically varied δ within the range $[0.1, 0.05, 0.01, 0.005, 0.001]$, representing a transition from higher worst-case privacy loss to lower levels. The resultant MI leakage, estimated by our estimator (MINE), between clients' input data and Split-1 layer activations is presented in Fig. 10. Interestingly, the variations in δ had negligible effects on average privacy loss, as δ primarily governs the

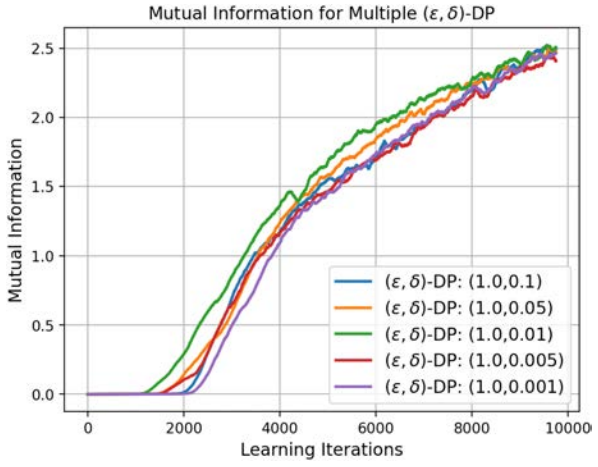


Fig. 10. The mutual information learning trend between input and (ϵ, δ)-DP protected outputs over fixed $\epsilon = 1.0$ and a range of δ values.

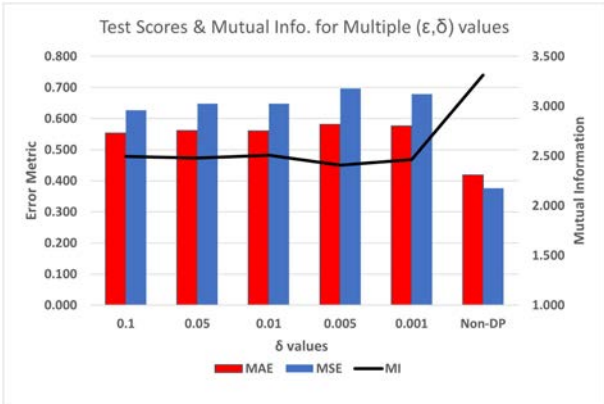


Fig. 11. The prediction test scores and mutual information values for models trained with (ϵ, δ)-DP over fixed $\epsilon = 1.0$ and a range of δ values.

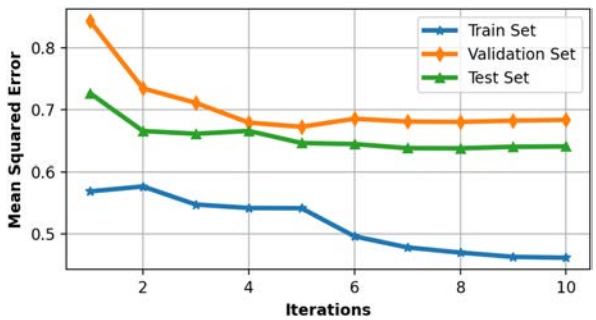


Fig. 12. The convergence graphs for train, validation, and test sets under $(\epsilon = 1.0, \delta = 0.005)$-DP setting.

worst-case scenario. Furthermore, Fig. 11 displays the model's prediction accuracy over the entire range of δ values. These metrics indicate that neither MAE nor MSE exhibit substantial variations within the considered range of δ values. The performance closely mirrors the outcomes observed when $\epsilon = 1.0$, as depicted in Fig. 9.

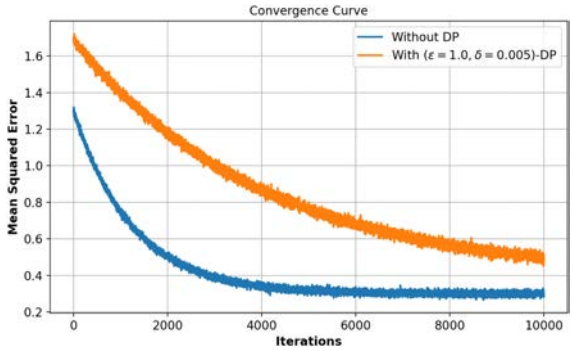


Fig. 13. The reconstruction MSE of the private client data by the FSHA on *SplitGlobal* model with and without DP.

To illustrate the impact of DP on convergence behavior, Fig. 12 presents the individual loss trends for the training, validation, and test sets over 10 epochs. The plot shows that validation and test losses decrease until about the 5th epoch and then stabilize, while the training loss continues to decline until around the 9th epoch. As expected, the overall loss floor in the DP setting is noticeably higher than in the non-DP case (Fig. 4) due to the added noise. Additionally, training time increased significantly, from roughly 200 minutes without DP to approximately 454 minutes with DP, representing a $2.54\times$ increase, primarily due to the computational overhead introduced by noise computations performed through the *Diffprivlib* library [43].

F. Feature Space Hijacking Attack

Although our framework is built under the 'honest-but-curious' assumption, i.e., the utility provider, which is in control of both GS and SP entities, does not intentionally attempt to reconstruct client data or poison the training process, we deliberately relax this assumption to evaluate the worst-case scenario. In this setup, the utility provider (or a compromised entity within it) can deviate from the standard training procedure and attempt to reconstruct private client input data from the shared Split-1 activations using the powerful Feature-Space Hijacking Attack (FSHA) [40].

To compare the effects of DP, we apply the FSHA attack on *SplitGlobal* model under two scenarios: with and without DP applied to the client side activations. This experiment uses the same settings as used in Section IV-E3. Under this scenario, the smoothed reconstruction MSEs achieved by the FSHA are illustrated in Fig. 13.

- *Without DP*, the attack model converges faster and achieves lower reconstruction MSE scores, consistent with observations from previous works [40], [46]. However, even in this case, our proposed framework maintains a reconstruction MSE floor around 0.305, indicating that the attacker still requires considerable effort to achieve meaningful reconstruction, partly due to the preserved complexity in the input-output relationship at Split-1.
- *With DP*, the attack becomes significantly more difficult. The attacker needs a much longer training period, and the resulting reconstruction accuracy is further degraded. This aligns with the intended purpose of DP: to degrade information leakage and protect against reconstruction-based attacks.

The above results are consistent with our analysis under Mutual Information (MI) estimation (Section IV-E1) as well. These results demonstrate that our proposed framework, especially when combined with differential privacy, exhibits improved robustness against powerful feature-space attacks.

G. Computational and Communication Overhead

In this section, we summarize our findings regarding the various overheads related to our training framework. For details, the reader is referred to the Appendix A, available online.

a) Communication Overhead: In our framework, the majority of communication overhead is handled between the GS and SP components, with minimal burden placed on the client. For each model update round, the client transmits activations from Split-1, amounting to approximately 6.3 MB, to the GS and receives a negligible 3 KB prediction output in return. During backpropagation, the client sends back loss gradients of the same size (3 KB), while the heavier 63 MB gradient tensors are exchanged between the GS and SP. As a result, the client is responsible for only about 4.55% of the total communication volume, highlighting the communication efficiency of our split learning design. Appendix A-A provides further numerical details, available online.

b) Computational Overhead: The computational workload in our split learning framework is unevenly distributed between the client (smart meter) and the utility provider (GS-SP). We approximated the number of multiplications for each model component, drawing from the FEDformer architecture. Our analysis shows that the smart meter (Split-1) is responsible for only about 12.66% of the total forward pass operations, while the remaining 87.34% are executed by the SP (Split-2). This is due to the client's role being limited to the initial encoding and decoding blocks, while the heavy attention and feed-forward layers are managed by the utility provider. Additionally, the backpropagation and model updates are entirely performed by the GS-SP entities. Since the backpropagation step generally involves around $2.5\times$ (average of $2-3\times$) the number of forward-pass operations, the client's total share of computations across a full model update round amounts to approximately 5%. This lightweight requirement affirms that smart meters can comfortably participate in training without a significant computational burden. A detailed breakdown is provided in Appendix A-B, available online.

c) Latency Overhead: Due to the architectural split, most of the computational operations are handled by the GS and SP, which collectively perform over 95% of the total workload. Consequently, the latency of each training iteration is primarily determined by the utility provider's infrastructure. Assuming higher-performance hardware at the GS-SP end, the total round-trip latency remains low. While our implementation was done on a single machine, Appendix A-C, available online, provides a theoretical latency analysis to offer real-world latency estimates under reasonable hardware assumptions.

d) Energy Overhead: The client device in our split learning setup is responsible for approximately 5% of the total computational workload during each model update. This offloading significantly reduces energy consumption at the edge, making the approach attractive for power-constrained devices like smart meters. As shown in Table II, our SplitPersonal model achieved an 18.5% improvement in prediction performance over the centralized baseline while shifting most of the energy burden to the utility provider's infrastructure. A deeper analysis of energy distribution and trade-offs is provided in Appendix A-D, available online.

H. Discussion

In Section IV, we used two SL strategies, *SplitPersonal* where we train neighbourhood-level personalized split networks, and *SplitGlobal* where a single global Split-2 is trained at SP for all personalized Split-1 models at GSs. Networks trained under both strategies achieved better or comparable performance compared to a centrally trained network, as seen in Table II. When predicting across neighbourhood clients, *SplitGlobal* model was able to get lower errors as compared to *SplitPersonal*, as shown in Table III, which is expected.

In Section IV-C, we tested the trained models on data from clients not used during the training stage. The results in Table IV show that the trained models performed well in this scenario; however, additional training using the new client's data improved performance. This is essential as new clients are constantly added to the system, and we might have very little data on them. Additionally, in Section IV-D, we compare the performance of models trained on the same clients versus training on random clients every epoch against unseen data and found that the models trained on random clients performed better. As with the availability of large amounts of smart meter data, training using all of it is often not feasible. Instead, training using a random subset of clients every epoch can lead to a model with good generalization capabilities. Furthermore, this model can be refined using unseen clients' data for added performance.

In Section IV-E, we analyzed the extent of privacy leakage arising from the sharing of clients' activations using MINE. In Fig. 7, we showed that even without the added noise, the Split-1 activations have significantly low MI w.r.t. the inputs due to their non-linear and complex relationship induced by the Split-1 model. Moreover, based on the estimated MI, a client can decide whether to forward the current batch activations to GS or not to mitigate privacy leakage. We further analyze the effects of introducing differential privacy as an additional layer of security on the model's performance. In Fig. 9, we see that with a moderate privacy budget of $\epsilon = 5.0$, the models performed similarly to the non-DP case and saw performance degradation only when $\epsilon \leq 2.5$, leading to strong privacy. With a trained model, the electricity service provider can perform individual-level predictions (requiring respective clients' involvement) and neighbourhood-level predictions using the cumulative load trend from the grid station servicing the neighbourhood.

In Section IV-F we evaluated the *SplitGlobal* framework under FSHA [40] in both non-DP and DP-enabled settings. In the non-DP case, the attacker's reconstruction achieved an MSE of 0.241, while enabling DP increased the reconstruction MSE to 0.328, indicating a clear improvement in privacy protection through DP regularization. Finally, in Section IV-G, our overhead analysis shows that the client-side in the Split learning framework contributes only approx. 5% of the total computational load and 4.55% of the communication load per training round. Additionally, under reasonable hardware assumptions, the client accounts for approximately 21% of the end-to-end latency and similar proportions in energy consumption, highlighting the lightweight nature of the client-side operations.

Our proposed split learning framework is well-suited for integration into modern power systems, particularly within infrastructures like Advanced Metering Infrastructure (AMI) and Energy Management Systems (EMS). The Split-1 module can be embedded in smart meters, while the GS and SP components can be deployed at the utility's backend, leveraging existing EMS or Distribution Management System (DMS) facilities. This decentralized architecture supports scalable and privacy-preserving model training across diverse consumer endpoints. However, practical deployment may face challenges such as communication latency, device heterogeneity, synchronization issues, and secure data exchange. Addressing these challenges will be essential to fully realize the framework's potential in real-world settings, and we identify these areas as important directions for future work.

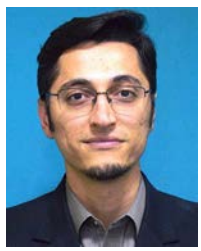
V. Conclusion

In this article, we propose a split learning framework to train a DL time series prediction model using the client smart meter data without compromising individual clients' privacy. Our proposed SL frameworks use the smart meters to perform a forward pass through the Split-1 network only, while the rest of the training is relegated to GS and SP entities. This ensures that the smart meter's main functionalities remain unhindered. Once trained, the energy provider retains the entire model, which can be used to perform load predictions for a single smart meter or the entire neighbourhood. The experimental results have shown that the performance of the trained models is better or on par with that of a centrally trained model. To analyze the extent of information leakage through the Split-1 network, we used mutual information neural estimation to approximate the MI between the input and output of the Split-1 network. The analysis showed that the MI leakage through the Split-1 network is limited. Furthermore, as an added layer of security, we analyzed the addition of ϵ-DP and (ϵ, δ)-DP to our framework under multiple privacy budgets. We found that the models performed similarly to the non-DP case under a medium privacy budget while observing low-performance degradation under relatively low privacy budgets.

References

[1] G. Notton and C. Voyant, "Forecasting of intermittent solar energy resource," in *Advances in Renewable Energies and Power Technologies*, Amsterdam, Netherlands: Elsevier, 2018, pp. 77–114.

[2] A. Arif, N. Javaid, M. Anwar, A. Naeem, H. Gul, and S. Fareed, "Electricity load and price forecasting using machine learning algorithms in smart grid: A survey," in *Proc. Workshops Int. Conf. Adv. Inf. Netw. Appl.*, Springer, 2020, pp. 471–483.
[3] N. Truonga, K. Suna, S. Wanga, F. Guittona, and Y. Guoa, "Privacy preservation in federated learning: Insights from the GDPR perspective," 2020. [Online]. Available: https://arxiv.org/abs/2011.05411
[4] A. Reinhardt, D. Burkhardt, M. Zaheer, and R. Steinmetz, "Electric appliance classification based on distributed high resolution current sensing," in *Proc. 37th Annu. IEEE Conf. Local Comput. Netw. Workshops*, 2012, pp. 999–1005.
[5] C. J. Hoofnagle, B. Van Der Sloot, and F. Z. Borgesius, "The European union general data protection regulation: What it is and what it means," *Inf. Commun. Technol. Law*, vol. 28, no. 1, pp. 65–98, 2019.
[6] K. Bonawitz et al., "Towards federated learning at scale: System design," *Proc. Mach. Learn. Syst.*, vol. 1, pp. 374–388, 2019.
[7] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv: 1812.00564*.
[8] L. Sehovac and K. Grolinger, "Deep learning for load forecasting: Sequence to sequence recurrent neural networks with attention," *IEEE Access*, vol. 8, pp. 36411–36426, 2020.
[9] Y. Tian, L. Sehovac, and K. Grolinger, "Similarity-based chained transfer learning for energy forecasting with Big Data," *IEEE Access*, vol. 7, pp. 139895–139908, 2019.
[10] M. N. Fekri, H. Patel, K. Grolinger, and V. Sharma, "Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network," *Appl. Energy*, vol. 282, 2021, Art. no. 116177.
[11] S. Ryu and Y. Yu, "Quantile-mixer: A novel deep learning approach for probabilistic short-term load forecasting," *IEEE Trans. Smart Grid*, vol. 15, no. 2, pp. 2237–2250, Mar. 2024.
[12] Y. Rao, W. Zhao, Z. Zhu, J. Lu, and J. Zhou, "Global filter networks for image classification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 980–993.
[13] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[14] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 22419–22430.
[15] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11106–11115.
[16] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 27268–27286.
[17] S. Li et al., "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5243–5253.
[18] I. Yazici, O. F. Beyca, and D. Delen, "Deep-learning-based short-term electricity load forecasting: A real case application," *Eng. Appl. Artif. Intell.*, vol. 109, 2022, Art. no. 104645. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197621004516
[19] A. Taïk and S. Cherkaoui, "Electrical load forecasting using edge computing and federated learning," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.
[20] J. Li, Y. Ren, S. Fang, K. Li, and M. Sun, "Federated learning-based ultra-short term load forecasting in power Internet of Things," in *Proc. IEEE Int. Conf. Energy Internet*, 2020, pp. 63–68.
[21] H. Liu, X. Zhang, X. Shen, and H. Sun, "A federated learning framework for smart grids: Securing power traces in collaborative learning," 2021, *arXiv:2103.11870*.
[22] M. N. Fekri, K. Grolinger, and S. Mir, "Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks," *Int. J. Elect. Power Energy Syst.*, vol. 137, 2022, Art. no. 107669.
[23] Y. Yang, Z. Wang, S. Zhao, and J. Wu, "An integrated federated learning algorithm for short-term load forecasting," *Electric Power Syst. Res.*, vol. 214, 2023, Art. no. 108830.
[24] Y. Liu, Z. Dong, B. Liu, Y. Xu, and Z. Ding, "FedForecast: A federated learning framework for short-term probabilistic individual load forecasting in smart grid," *Int. J. Elect. Power Energy Syst.*, vol. 152, 2023, Art. no. 109172. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061523002296
[25] M. A. Husnoo, A. Anwar, N. Hosseinzadeh, S. N. Islam, A. N. Mahmood, and R. Doss, "A secure federated learning framework for residential short term load forecasting," *IEEE Trans. Smart Grid*, vol. 15, no. 2, pp. 2044–2055, Mar. 2024.
[26] H. Liu, X. Zhang, H. Sun, and M. Shahidehpour, "Boosted multi-task learning for inter-district collaborative load forecasting," *IEEE Trans. Smart Grid*, vol. 15, no. 1, pp. 973–986, Jan. 2024.
[27] Y. Sakuma and H. Nishi, "Hierarchical multiobjective distributed deep learning for residential short-term electric load forecasting," *IEEE Access*, vol. 10, pp. 69950–69962, 2022.
[28] T. Wang, C. Ren, D. Z. Yang, and C. Yip, "Domain-adaptive clustered federated transfer learning for EV charging demand forecasting," *IEEE Trans. Power Syst.*, vol. 40, no. 2, pp. 1241–1254, Mar. 2025.
[29] C. Feng, D. Feng, G. Huang, Z. Liu, Z. Wang, and X.-G. Xia, "Robust privacy-preserving recommendation systems driven by multimodal federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 36, no. 5, pp. 8896–8910, May 2025.
[30] S. Guo, A. Zhang, Y. Wang, C. Feng, and T. Q. S. Quek, "Semantic-enabled 6G communication: A task-oriented and privacy-preserving perspective," *IEEE Netw.*, 2025, doi: 10.1109/MNET.2025.3547760.
[31] S. Abuadbba et al., "Can we use split learning on 1D CNN models for privacy preserving training," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 305–318.
[32] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3/4, pp. 211–407, 2014.
[33] L. Jiang, Y. Wang, W. Zheng, C. Jin, Z. Li, and S. G. Teo, "LSTMSPLIT: Effective split learning based LSTM on sequential time-series data," 2022, *arXiv:2203.04305*.
[34] M. I. Belghazi et al., "MINE: Mutual information neural estimation," 2018, *arXiv: 1801.04062*.
[35] M. Abadi et al., "Deep learning with differential privacy," in *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[36] E. Erdogan, A. Kupcu, and A. E. Cicek, "SplitGuard: Detecting and mitigating training-hijacking attacks in split learning," 2021, *arXiv:2108.09052*.
[37] A. Paverd, M. Andrew, and B. Ian, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," University of Oxford, Tech. Rep., 2014. [Online]. Available: https://www.cs.ox.ac.uk/people/andrew.paverd/casper/casper-privacy-report.pdf
[38] Z. Ji, Z. C. Lipton, and C. Elkan, "Differential privacy and machine learning: A survey and review," 2014, *arXiv:1412.7584*.
[39] A. De, "Lower bounds in differential privacy," in *Proc. 9th Theory Cryptogr. Conf.*, Taormina, Sicily, Italy, Springer, 2012, pp. 321–338.
[40] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2113–2129.
[41] M. D. Donsker and S. S. Varadhan, "Asymptotic evaluation of certain Markov process expectations for large time. IV," *Commun. Pure Appl. Math.*, vol. 36, no. 2, pp. 183–212, 1983.
[42] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[43] N. Holohan, S. Braghin, P. Mac Aonghusa, and K. Levacher, "Diffprivlib: The IBM differential privacy library," 2019, *arXiv: 1907.02444*.
[44] Implementation code of split load forecasting for smart grid, 2024. [Online]. Available: https://github.com/AsifIqbal8739/SplitLoadForecasting
[45] Y. Liu, X. Zhu, J. Wang, and J. Xiao, "A quantitative metric for privacy leakage in federated learning," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 3065–3069.
[46] G. Gawron and P. Stubbings, "Feature space hijacking attacks against differentially private split learning," 2022, *arXiv:2201.04018*.



Asif Iqbal received the BS degree in telecommunication engineering from NUCES-FAST, Peshawar, Pakistan, the MS degree in wireless communications from LTH, Lunds University, Sweden, and the PhD degree in electrical & electronics engineering from The University of Melbourne, Melbourne, Australia, in 2008, 2011, and 2019 respectively. He is currently working as a research fellow with the Department of Electrical & Computer Engineering, National University of Singapore, Singapore. He previously served on the faculty of NUCES-FAST, Peshawar, Pakistan as an assistant professor. His research interests include signal processing, deep learning, sparse signal representations, and privacy-preserving machine learning.

Prosanta Gope (Senior Member, IEEE) is currently working as an associate professor with the Department of Computer Science (Cyber Security), University of Sheffield, U.K. He served as a research fellow with the Department of Computer Science, National University of Singapore (NUS). Primarily driven by tackling challenging real-world security problems, he has expertise in Lightweight Authentication, Authenticated Encryption, 5G and Next Generation Communication Security, Privacy-Preserving Machine Learning, Security in the Internet of Things, Smart-Grid Security, PUF-based security system and IoT Hardware. He has authored more than 100 peer-reviewed articles in several reputable international journals and conferences and has four filed patents. Several of his papers have been published in high-impact journals (such as *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE/ACM Transactions on Networking*), and prominent security conferences (such as IEEE S&P, ACM CCS, IEEE Computer Security Foundations Symposium (CSF), Privacy Enhancing Technologies Symposium (PETS), ESORICS, Euro S&P, IEEE TrustCom, IEEE HoST, etc.) He has been a TPC member and co-chair in several reputable international conferences, including PETS, ESORICS, IEEE TrustCom, IEEE GLOBECOM (Security Track), and ARES. He currently serves as an associate editor of the *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information & Forensics Security*, *IEEE Transactions on Services Computing*, *IEEE Systems Journal*, and the *Journal of Information Security and Applications* (Elsevier). His research has been funded by EPSRC, Innovate U.K., and the Royal Society.

Biplab Sikdar (Senior Member, IEEE) received the BTech degree in electronics and communication engineering from North Eastern Hill University, Shillong, India, in 1996, and the MTech degree in electrical engineering from the Indian Institute of Technology Kanpur, Kanpur, India, in 1998, and the PhD degree in electrical engineering from the Rensselaer Polytechnic Institute, Troy, New York, in 2001. He was a faculty member with the Rensselaer Polytechnic Institute from 2001 to 2013 and was an assistant professor and associate professor. He is currently a professor and head of the Department of Electrical and Computer Engineering, National University of Singapore. He also serves as the director of the Cisco-NUS Corporate Research Laboratory. His current research interests include wireless networks and security for the Internet of Things and cyber-physical systems. He has served as an associate editor for the *IEEE Transactions on Communications*, *IEEE Transactions on Mobile Computing*, *IEEE Internet of Things Journal* and *IEEE Open Journal of Vehicular Technology*.